♦ Policy-Based Network Load Management

Ashfaq Hossain, Houshing F. Shu, Charles R. Gasman, and Randolph A. Royer

Current solutions for balancing load among Internet servers are based on the use of network address translation (NAT) and virtual Internet protocol (IP) addresses. In this work, we describe the design and implementation of an intelligent load-management prototype that is not based on the traditional approaches mentioned above, but can be efficiently built into existing high-end switch offerings from Lucent Technologies, such as the Cajun™ P550™ Gigabit Routing Switch. The key features of this prototype are (1) IP packets are not modified in any way—only Ethernet addresses are manipulated; (2) each machine that is a member of a server farm uses a shared IP address, in addition to unique IP addresses; and (3) a sophisticated tunable algorithm is used to select a server for the next incoming request.

Introduction

Policy-based network management tries to improve the services available over the Internet by intelligently configuring network devices (such as switches) and the software executing on them using some well-defined policies (rules). These rules can be locally developed or they can be downloaded from a central database via policy servers using protocols promoted by Lucent Technologies[1] such as common open policy service (COPS), DIAMETER, and lightweight directory access protocol (LDAP). To achieve this goal of improved quality of service (QoS), the concept of *differentiated IP services* has recently been introduced with expedited forwarding and assured forwarding traffic classes.[2]

This paper describes the rigorous implementation of an intelligent policy-based network switch management technique that results in improved QoS for Internet applications. The policy, which can be incorporated into existing high-end data-switch offerings from Lucent, can either be initialized with local configuration information or be downloaded from a policy server. Using this policy, our switch intelligently manages incoming Internet service requests among a group of servers, resulting in much better QoS for the requested service.

We provide a brief background of the overall problem and the available solutions in "Background of Policy-Based Server Load Management." We then describe a flexible switch architecture on which our own solution is based in "Switch Overview." We present our policy-based load-management algorithm in "Server Selection Algorithm," give details of our prototype in "Overview of the Prototype," and then explain our experiments in "Experimental Runs."

Background of Policy-Based Server Load Management

The problem of managing load is deceivingly simple and appears in almost all types of technology solution offerings.[3,4] Essentially, it is a contention for limited resources. From the perspective of providing service over the Internet (for example, hypertext transfer protocol [HTTP], file transfer protocol [FTP], and telnet), a major problem is contention for the server resources that promise and provide these services. The delay perceived by an end user (client) in the Internet is a function of server response delay and network delay (for example, from routers and switches), where the latter is the more prominent bottleneck in many cases. Still, a server's response time to a request may become unacceptably high, far exceeding network propagation and congestion delays, when

servicing a large number of incoming requests.

The solution to the contention problem, therefore, is to add more servers in a group to share the request load. Such a group is being called a *server farm*. Since servers in the Internet are required to have unique IP and media access control (MAC) (that is, Ethernet) addresses, this solution itself is a paradox: the servers maintain their unique IP and MAC addresses while presenting to the clients a single (advertised) IP address for the server farm as a whole. A load-management switch, therefore, has to receive all the incoming requests to the advertised IP address from the Internet (the clients) and then, according to some policy, assign a server from the farm to respond to each request and to route all the packets between this server and the client for the entire length of the IP connection. This concept is shown in **Figure 1.**

The responsibility of the switch for managing load is handled in currently available solutions as follows:

- *Server selection.* Servers are selected by round robin according to weighted criteria such as response time, number of active connections, and particular type of request.
- *Packet routing between selected farm server and client.* TCP/UDP/IP packet headers are changed in the switch for packets going in either direction. This requires costly recalculation of checksums at different layers that can make the switch itself the bottleneck.

Solutions to such bottlenecks and to similar Internet problems have been proposed and, in some cases, made commercially available. A load-balanced Web access system has been proposed[5] that uses redirection servers for redirecting Web requests; this implies regenerating new TCP/IP packets carrying modified HTTP requests from the redirection servers to the back-end servers. Several techniques for hosting Internet service on a cluster of servers are discussed by Damani et al.[6] Kumar, Lakshman, and Stiliadis[7] have discussed router architectures for providing customer-specific differentiated services in very-high-speed networks. Currently available commercial systems include Alteon 700 Series Switches and the ACEdirector Switch[8] from Alteon WebSystems, as well as the CS-800 and CS-100 Content Smart* Web Switches from ArrowPoint Communications.[9] The Alteon switches perform address translation at the TCP/IP and Ethernet layers, thereby requiring dedicated hardware for fast regeneration of TCP packets and the associated checksums.[8]

With the responsibilities of the switch in perspective, this paper contributes originally in the following ways:

- An algorithm has been developed for server selection considering several important factors. The algorithm defines an *objective function*, which is a measure of the total load differences among all the servers in the farm. The factors in the function are weighted to suit the selected policy. If there are hard limitations mandated by the policy, they are taken into

Panel 1. Abbreviations, Acronyms, and Terms

API—application program interface
COPS—common open policy service
CPU—central processing unit
DIAMETER—extension of remote authentication dial-in user service (RADIUS)
FP—fast path
FPT—fast path table
FTP—file transfer protocol
HTML—HyperText Markup Language
HTTP—hypertext transport protocol
HTTPS—hypertext transport protocol secure
IEEE—Institute of Electrical and Electronics Engineers
IP—Internet protocol
LAN—local area network
LDAP—lightweight directory access protocol
MAC—media access control
NAT—network address translation
NIC—network interface card
OS—operating system
PCI—peripheral component interconnect
ping—packet Internet groper; an Internet utility to determine whether an IP address is online
POSIX—portable operating system interface for UNIX;* an IEEE standard
QoS—quality of service
SP—slow path
SPT—slow path table
TCP—transmission control protocol
UDP—user datagram protocol

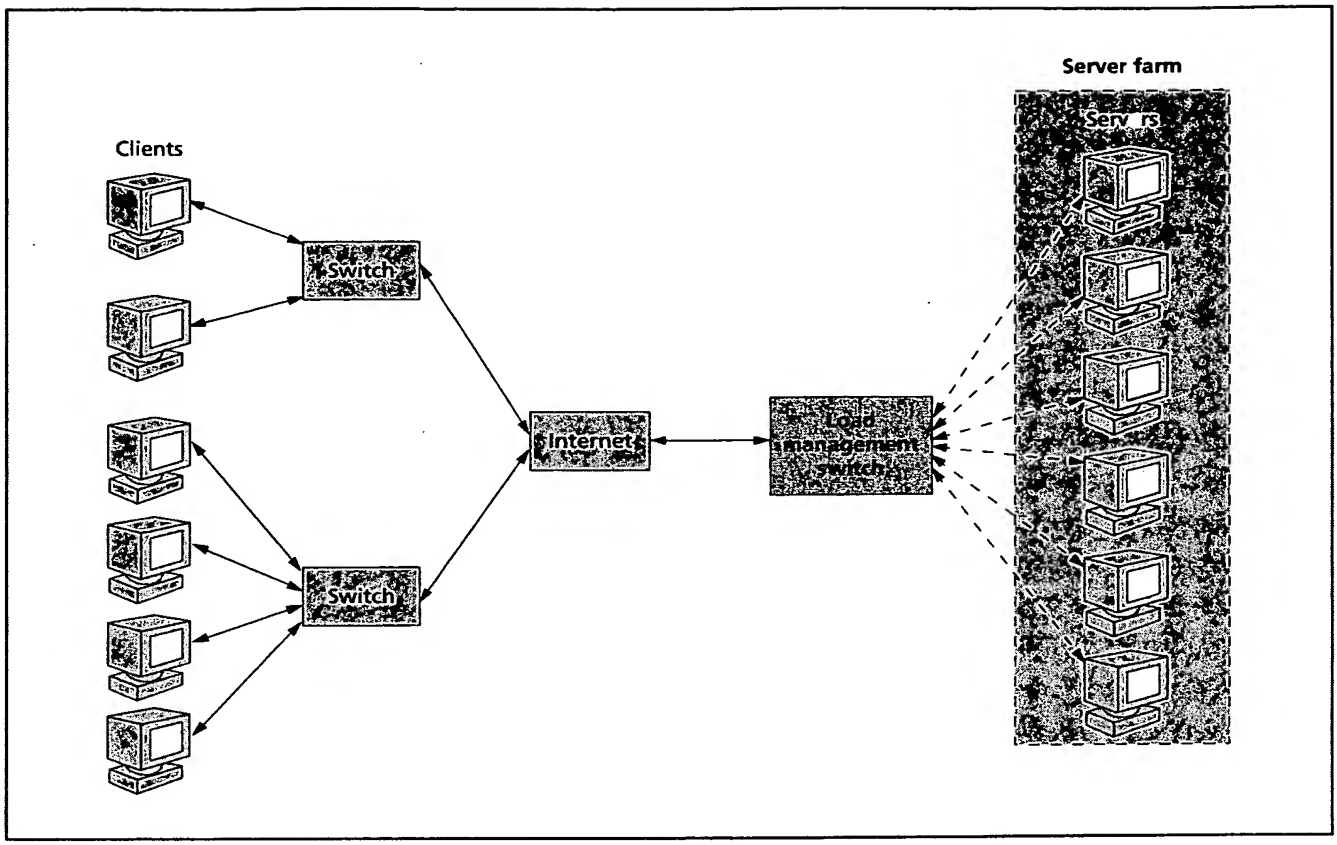

Figure 1.
Concept of load management with a server farm in the Internet.

consideration in the form of constraints in the optimization algorithm. This is explained in detail under "Server Selection Algorithm."

- A fully functional prototype has been developed to demonstrate the interworkings of server selection and packet address translation. The work presented in this paper differs from the above-mentioned solutions in one major way—our prototype switch does not modify the TCP/UDP/IP packets at all. Address translation is performed only at the MAC layer and load balancing is still achieved. A special feature available in modern operating systems, namely, *IP aliasing,* was used successfully in our prototype development to achieve this objective. This is further explained in "Overview of the Prototype."

Switch Overview

Our prototype, which incorporates a custom selection algorithm, has been developed with a commercially available high-end switch in perspective—namely, Lucent's Cajun™ P550™ Gigabit Routing Switch for Gigabit Ethernet. Although the load-management concepts demonstrated in our work can also be applied to other switches quite easily, the Cajun P550 switch was chosen as our baseline because it is among the most advanced Ethernet data switch offerings from Lucent at this time.

We now describe some of the key characteristics of a P550-like switch that relate to the development of our prototype. These aspects of the switch are shown schematically in **Figure 2.**

Fast Path

The switch maintains a *fast path table (FPT).* The rows of this table contain routing information for

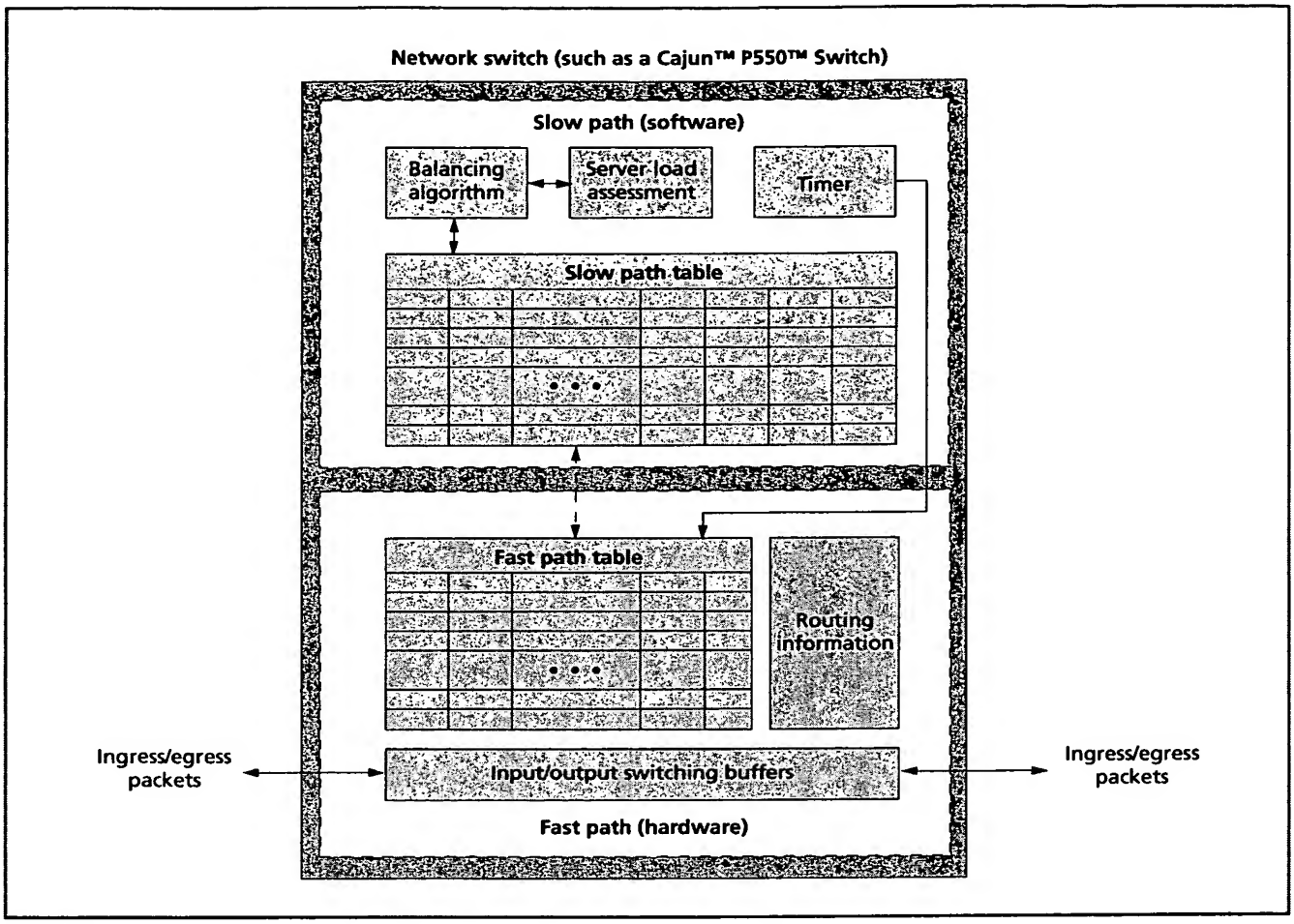

Figure 2.
Switch functionalities related to load-management prototype.

already established flows. A timer field is associated with each row and is initialized when the row is written. After the timer expires, information can be retrieved from the *slow path table (SPT)*, which is described below. The *fast path (FP)* is implemented entirely in hardware to be very fast. The entries in the FPT are established by firmware in the *slow path (SP)*.

As a new packet arrives, the FPT is searched for routing information. The keys (some header fields) used for the FPT search are derived by the hardware from the newly arrived packet. A match indicates that routing information already exists for this connection, and the packet is forwarded to the intended egress port. The timer for this entry in the FPT is refreshed at this point. For a non-match, the packet is handed over to the SP firmware.

Slow Path

The SPT is a superset of the FPT—it contains all the information stored in the FPT (and more) that is related to a particular flow. Just like the FPT, each SPT entry has a timer field that is initialized when the row is written. The expiry interval in the SPT is a function of the type of connection (such as FTP, HTTP, or secure HTTP [HTTPS]). The expiry is set at a slightly higher value than the allowed dormancy period for the connection. When a packet is received in the SP, the SPT is searched with keys extracted from the newly arrived packet. If a match is found, it indicates that the corresponding FPT entry has been purged due to its timer expiry. The SP then re-establishes the entry in the FP and allows the packet to be routed by the hardware. The timer entry in the

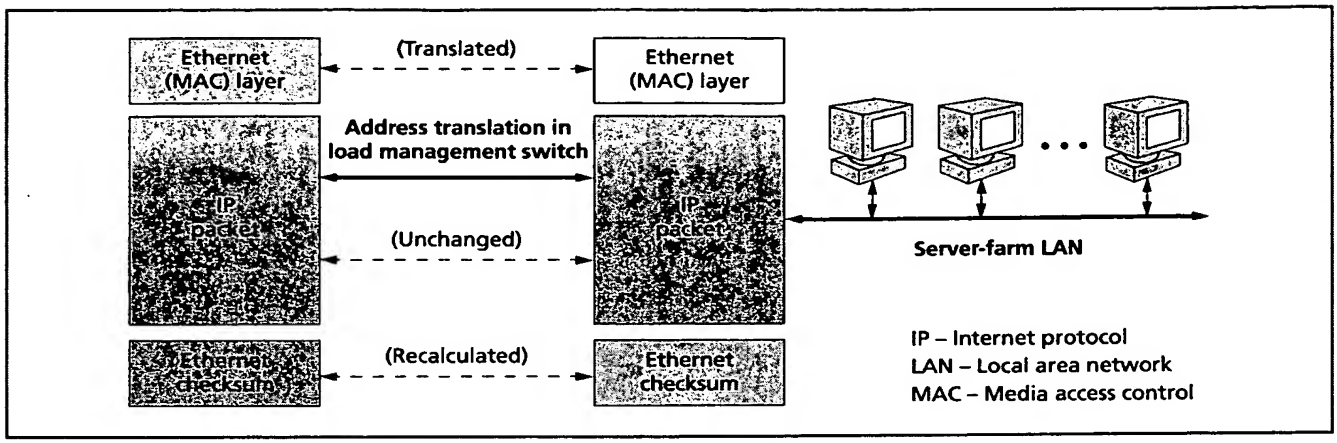

Figure 3.
Address translation in load-management prototype.

SPT for this connection is also refreshed at this point.

If a match is not found in the SPT, this indicates either that the packet requires an entirely new connection or that the SPT entry has been purged due to SP timer expiry. For both cases, the SP generates routing information using a suitable method. It then creates entries in the SPT as well as the FPT. The packet is then routed accordingly by the hardware.

It can be noted here that, if the SPT entry has been purged due to a long dormancy period of the connection (thereby, timer expiry), the server selected from the farm by the load-balancing algorithm may not be the one that was originally selected for this client. In this situation, the server will reject the connection automatically, forcing the client to re-initiate the session. The entries in the SPT and FPT will eventually expire for the rejected connection.

As shown in Figure 2, the SP is also responsible for load assessment statistics that are used by the balancing algorithm for server selection. The timer in the SP is responsible for establishing and purging timing fields in the FP.

Traffic Redirection and Address Translation for Load Management

The switch "listens" to a set of advertised IP address/port-number combinations. From the load-management perspective, these are the addresses for which balancing is provided. The servers, which provide services to these advertised address combinations, are grouped together in a farm and connected either to the switch in a local-area-network (LAN) configuration or directly to the ports of the switch. Each server in the farm has a unique IP address. In addition, each server listens to the advertised (common) IP address; this is referred to as *IP aliasing*. When an IP packet arrives at the server with the aliased IP as its destination host, the server will receive the packet if the Ethernet frame carrying the IP packet contains the correct MAC address of the server's network interface card (NIC). Therefore, the responsibility of the switch is to modify the Ethernet source and destination MAC addresses of the aliased IP packets and then transmit them onwards so they can be picked up by the intended server in the farm. The switch performs this MAC address translation after the selection algorithm selects a particular server from the farm. For the current stage of our prototype, we consider that the MAC addresses of the servers are known to the switch. Similar translations on MAC addresses are performed in the reverse direction as well.

The concept of MAC address translation by the switch is shown in **Figure 3.**

Server Selection Algorithm

In order to optimize the decision made by the system, we use an objective function that provides a highly tunable weighted selection capability. The objective function uses several data points from each

server as collected by the system. They are

- Number of open sessions (TCP or UDP);
- Time taken by the server for opening a new connection (measured between switch and server);
- Number of bytes delivered over a certain time period; and
- Number of new session requests assigned to the server over a certain time period.

These statistics are constantly collected and maintained by the traffic statistics module for each of the servers being balanced in the farm. For each data point, the current value for a given server is compared against the statistic across the set of farm servers and multiplied by a weighting factor.

The weighting coefficients for each term in the objective function can be tuned to work with specific traffic types (for example, streaming video), altering the importance of a particular term in the function for different types of traffic. In this way, our system can keep a set of objective functions tuned for specialized traffic sessions, in addition to an objective function to handle generic data sessions such as HTTP. The terms used in the objective function are defined in **Panel 2.**

With the factors for load management now defined, an objective function is formulated taking them into consideration. The weighting factors, $C1$, $C2$, $C3$, and $C4$, are chosen such that the selection index, $SelIndex_j$, is a dimensionless quantity.

$$
\begin{aligned}
SelIndex_j = {} & C1 \times \sum_{k \neq j} |(OpenConnTime_j \\
& + AvgOpenConnTime - OpenConnTime_k)| \\
& + C1 \times \sum_{p \neq q \neq j} |(OpenConnTime_p - OpenConnTime_q)| \\
& + C2 \times \sum_{k \neq j} |(NumBytes_j + AvgNumBytes - NumBytes_k)| \\
& + C2 \times \sum_{p \neq q \neq j} |(NumBytes_p - NumBytes_q)| \\
& + C3 \times \sum_{k \neq j} |(NumReqs_j + AvgNumReqs - NumReqs_k)| \\
& + C3 \times \sum_{p \neq q \neq j} |(NumReqs_p - NumReqs_q)| \\
& + C4 \times \sum_{k \neq j} |(NumConns_j + 1 - NumConns_k)| \\
& + C4 \times \sum_{p \neq q \neq j} |(NumConns_p - NumConns_q)|
\end{aligned}
$$

Panel 2. Terms Used in Objective Function

AvgNumBytes—average number of bytes served by the server group during last time quantum
AvgNumReqs—average number of new requests served by the server group during last time quantum
AvgOpenConnTime—average time taken by the server to set up a new connection during last time quantum
j—index of farm server under consideration
k—index of *other* servers in farm
$NumBytes_k$—number of bytes served from server *k* during last time quantum
$NumConns_k$—number of open TCP/UDP connections on server *k*
$NumReqs_k$—number of service requests assigned to *k* during last time quantum
$OpenConnTime_k$—time to open a new connection to server *k*
p—index of server in farm not equal to *j*
q—index of another server in farm not equal to *p* or *j*
$SelIndex_j$—selection index for server *j*

In order to reduce computational complexity in the load-management module, the *SelIndex* parameter is calculated at the beginning of a time quantum. The value of the time quantum is itself an adjustable quantity depending on the load on the switch. For all new requests during the current time quantum, the server with the smallest *SelIndex* is selected. It is conceivable that *SelIndex* could be calculated for every new incoming request, providing more granularity in load management; however, we have not considered that in our prototype development.

For systems with less statistics processing power, the number of different factors used in the calculation can be reduced to produce the more common weighted round-robin capability offered in many switches today capable of balancing load.

Overview of the Prototype

In this section, we describe the implementation of our prototype. We have used the Linux* operating system (OS) platform for all code development. This choice has been influenced by the recent popular shift in the industry towards open-systems development.

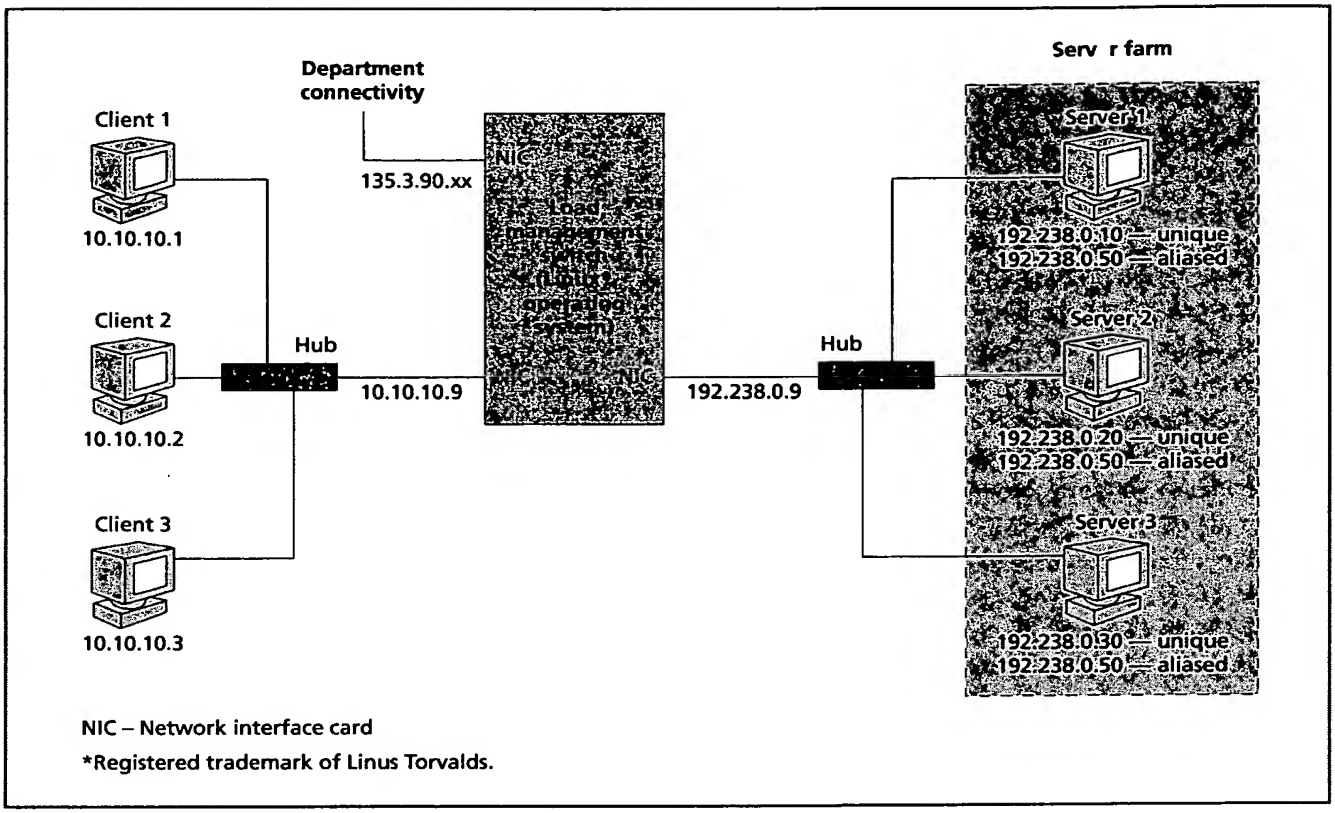

Figure 4.
Laboratory configuration for load-management prototype development.

Laboratory Configuration

The laboratory configuration of our prototype is shown in **Figure 4**. The prototype code executes in the load-management switch (the Linux machine), which is equipped with three peripheral-component-interconnect (PCI)–based 10/100 Mb/s NICs. The first NIC of this switch is connected to the department's LAN (`135.3.90.xx`) and is only used for maintaining remote connectivity to this machine; it is not related to the interworking of the load-management prototype. The second NIC (`10.10.10.9`) of the switch is connected to a set of clients through a hub. This group of clients emulates the real-world scenario; network service requests are generated from these clients to the advertised IP address/port-number combinations. In our prototype, we have used clients running Solaris* and Linux operating environment software—all having unique IP addresses. This second NIC and the group of clients constitute the `10.10.10.xx` network for our experiments. The third NIC in the switch is connected to a group of three servers that constitute the server farm. These servers and the third NIC on the switch constitute the `192.238.0.xx` network. Two of the servers used are based on the Solaris system and one is based on the IRIX* system.

It is important to note that, apart from having unique IP addresses in the `192.238.0.xx` private network, the servers all share one or more aliased IP addresses. These aliased IP addresses are the ones advertised by the switch. Packets sent to these IP addresses are forwarded to the selected server by the switch after the proper MAC address changes (described in "Switch Overview"), thereby allowing each member in the server group to receive them uniquely.

Interworking of the Prototype

Our prototype code uses the POSIX-compliant multithreaded approach. The use of threads, instead of processes, has helped to modularize the types of

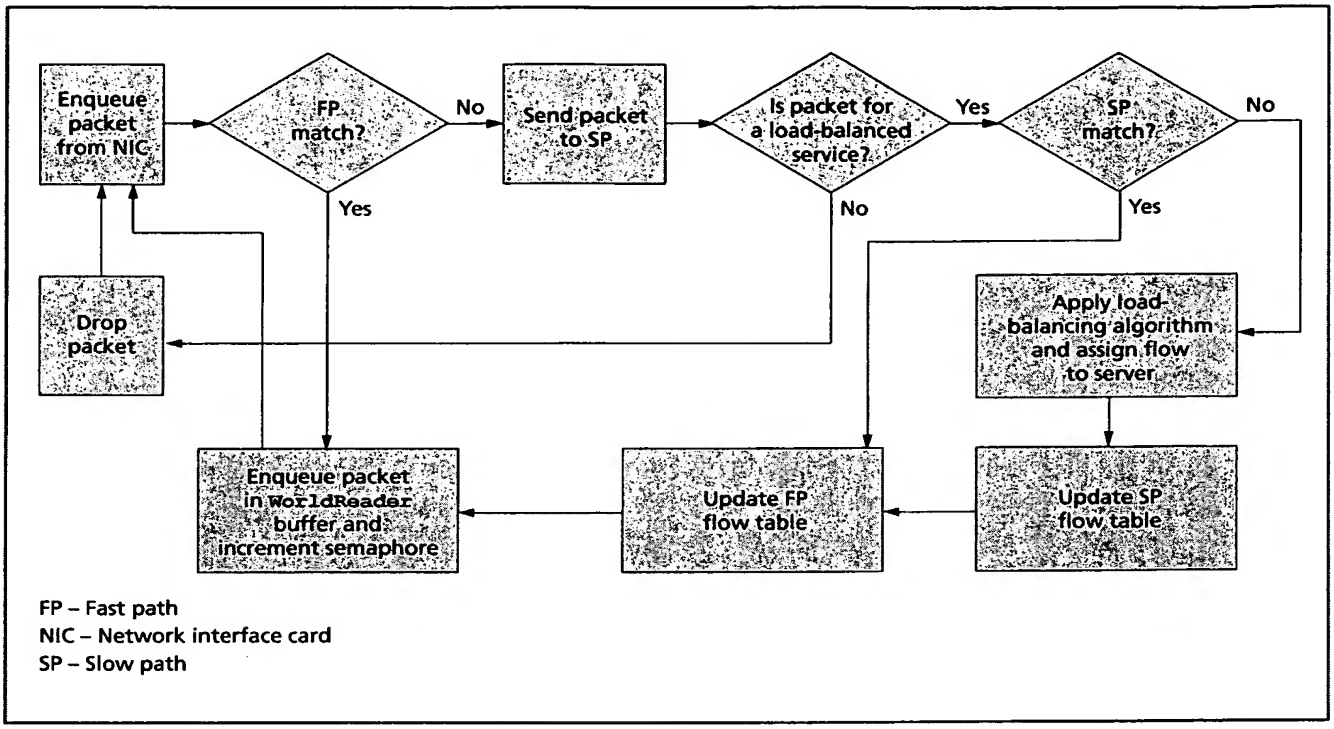

Figure 5.
WorldReader() thread of the prototype.

activities with minimum overhead to the kernel resources, leaving more central processing unit (CPU) cycles for the user-level code execution. Synchronization among threads has been achieved using semaphores, and access to shared data is protected using mutex thread locks.

The SOCK_PACKET feature available in the current Linux release (Red Hat* Linux 6.0) has been used very conveniently in our prototype development. If this parameter is set in the `socket()` call, the Linux OS allows the user-level application to receive Ethernet frames. This aspect of having complete access to Ethernet frames without any modification to the kernel has been helpful to our rapid prototype development. This feature also allows the Ethernet frames, even if they are modified, to be written out of the NICs using the commonly used `send()` and `write()` application program interfaces (APIs). Most importantly, the Ethernet checksum of the modified packet is calculated by the NIC hardware—*not* by the user-level application.

We now explain the threads we have used in our prototype development.

`WorldReader()` thread. This thread reads the packets coming in from the world. In our laboratory scenario, these are packets generated by the client group that may be initial connection-request packets or packets from an already established connection. The NIC in our emulated switch connected to the client group runs in promiscuous mode. This mode, in conjunction with the SOCK_PACKET option described above, allows all packets in the Ethernet to be seen by the load-management software executing at the user level.

The detailed operation of this thread is shown in **Figure 5** and is described in the following steps:

1. An Ethernet frame is received from the NIC.
2. The FPT is searched for the matching keys of this newly arrived packet. If a match is found in the FPT, it is an already established connection. This also means that a server has been selected for this connection earlier. The MAC (Ethernet) destina-

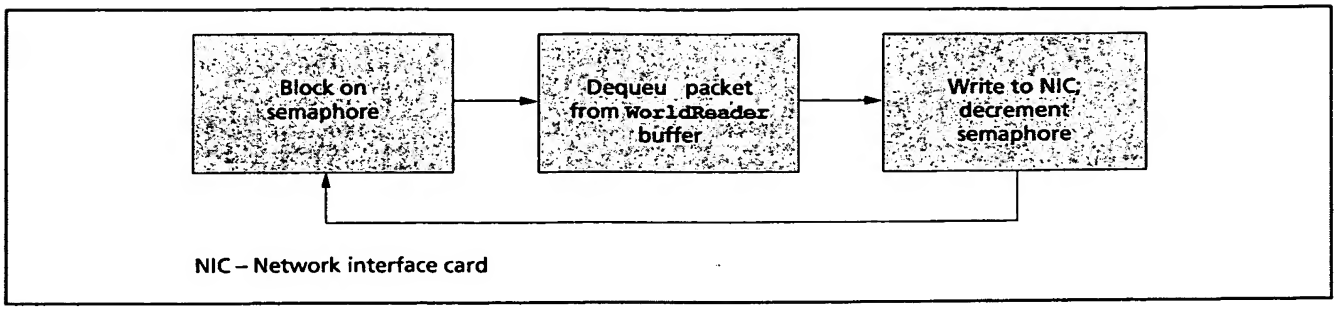

Figure 6.
`FarmWriter()` thread of the prototype.

tion address of this frame is changed to the server NIC's MAC address. The source MAC address is changed to the outgoing NIC's MAC address. No other field of the entire packet is changed. The MAC address-translated Ethernet frame is enqueued in the `WorldReader` buffer to be dequeued and transmitted by the `FarmWriter()` thread. The Ethernet checksum is not calculated by the load-management code. After enqueuing the frame in the `WorldReader` buffer, a semaphore is incremented to signal the `FarmWriter()` (consumer) thread of the availability of this frame.

3. If there is not an FPT match, then the frame is either a packet for an entirely new connection or a packet for a connection whose FPT entry has been purged by the FP timer. This newly arrived Ethernet frame is then sent to the SP. In the SP, it is first checked whether the destination IP address and TCP/UDP port of the received packet match with one of the advertised IP address/port-number combinations. For a mismatch, the packet is discarded. For a match, the SPT is searched for the matching keys of this newly arrived packet. A match in the SPT indicates that this packet is from an earlier connection whose corresponding FPT entry has been purged by the FP timer. A mismatch means the packet is from an entirely new connection. A server needs to be selected for this request. The load-balancing algorithm is now invoked to select the server.
4. The server selection criteria have been explained in detail in the section "Server Selection Algorithm." Once invoked by the SP, the load-balancing module returns the Ethernet MAC address of the selected server. An SPT entry is created, and a corresponding FPT entry is created. The Ethernet MAC addresses are modified as explained above. The packet is enqueued in the `WorldReader` buffer and the semaphore is incremented to signal the `FarmWriter()` thread.
5. Another packet for this same connection now encounters an FPT match and is enqueued in the `WorldReader` buffer immediately. In case the connection is dormant, the FPT entry is purged, the packet is sent to the SP, and there should be an SPT match.

`FarmWriter()` thread. If a packet is available in the `WorldReader` buffer, the `FarmWriter()` thread dequeues it and writes it out to the `192.238.0.xx` NIC by calling a `send()` function. The Ethernet checksum of the modified frame is calculated by the NIC hardware. The semaphore is decremented to indicate transmission of a packet. The operation of this thread is shown in **Figure 6**.

`FarmReader()` thread. This thread is responsible for collecting response packets from the server group and enqueuing them into the `FarmToWorld` buffer appropriately. It is similar in operation to the `WorldReader()` thread, except that no server selection is involved. The detailed operation of the `FarmReader()` thread is shown in **Figure 7** and is described below:

1. An Ethernet frame is received from the NIC in the reverse direction (server farm to the world).
2. The FPT is searched with the keys of this packet. If a match is found, the source and destination MAC addresses are changed in the Ethernet

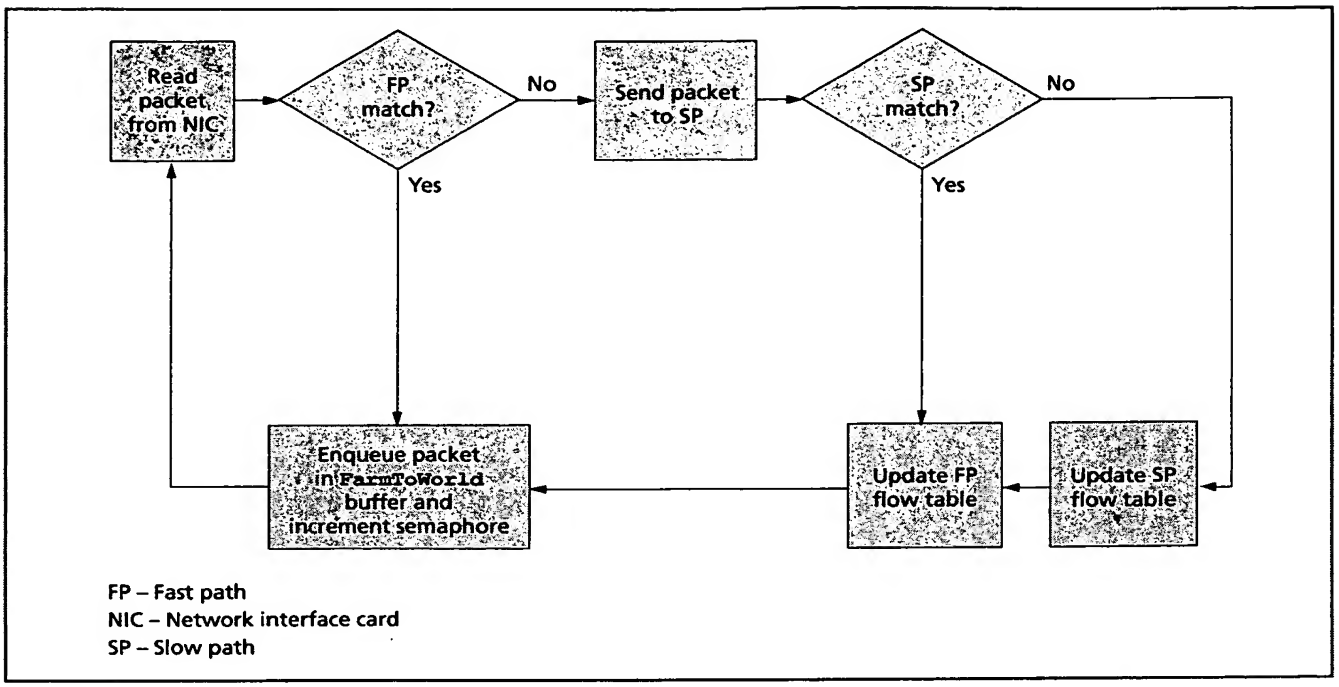

Figure 7.
FarmReader() thread of the prototype.

frame appropriately. The frame is then enqueued in the `FarmToWorld` buffer. A semaphore is incremented to signal the `WorldWriter()` thread about the availability of this packet.

3. If an FPT match is not found, the packet is sent to the SP. As in the forward direction, the packet is either the initial response from the server, or the corresponding FPT entry has been purged due to a long dormancy period. The SPT is searched; if a match is not found in the SPT, it is an initial response. An SP entry and an FP entry are created, and the MAC addresses are changed. The packet is then enqueued in the `FarmToWorld` buffer, and the semaphore is incremented.
4. If a match is found in the SPT, the corresponding FPT entry is re-established and the packet is enqueued in the buffer after its MAC addresses are changed.

`WorldWriter()` thread. If a packet is available in the `FarmToWorld` buffer, the `WorldWriter()` thread dequeues it and writes it out to the `10.10.10.xx` NIC by calling a `send()` function. The Ethernet checksum of the modified frame is calculated by the NIC hardware. The semaphore is decremented to indicate transmission of a packet. The flow diagram of this thread is similar to that of the `FarmWriter()` thread.

`TimerPolice()` thread. This thread, whose function is illustrated in **Figure 8**, is responsible for maintaining the timer functions in the FPT and the SPT. Each row of these tables has a timer field. These fields are initialized when entries are created in the FPT and the SPT corresponding to a new connection. The `TimerPolice()` thread refreshes the timer fields each time there is a hit during an FPT search or an SPT search. If the timer expires, the full row is purged by the `TimerPolice()` thread.

Statistics Collection

The `ServerStatus()` thread is responsible for maintaining status information for the servers in the farm. The statistics collected by this thread are used by the load-balancing algorithm (see "Server Selection Algorithm") to select a candidate server for the next new incoming request. A statistics table is maintained where each row shows the statistics of each server

collected over the last time quantum. The pieces of information collected from each server are shown in **Figure 9**. *Number of active connections* indicates, as the name suggests, the total number of connections being serviced by the server at the end of the last time quantum. This does not give any indication of the type or persistence of the connections. *Delivered throughput* is the total amount of data bytes transferred to the clients by the server during the last time quantum. This provides an idea of the load on the server, while *connection type* indicates the number of different types of connections to the server (for example, FTP, HTTP, and HTTPS). *Response time* is the time taken by a server to actually respond to a new incoming request—that is, the time to send the first packet in the reverse direction.

The `ServerStatus()` thread is also responsible for identifying any downed servers in the farm and for eliminating them from being selected for future incoming requests until the situation is corrected. This can be achieved by simply waiting on high threshold time after making some request (for example, an HTML request). Although a non-response for this request may indicate that the daemon is out of service, the switch may then send simple packet-Internet-groper (`ping`) requests to check whether that particular server is alive or not.

Experimental Runs

We have performed several experimental runs to validate our approach to load management. We have generated different types of service requests addressed to the advertised IP address/port-number combinations and have verified that the responses have originated from the different servers in the farm as they were selected by the load-balancing algorithm. The timer expiry in the FPT for dormant connections and reestablishment of the FPT entries from the SPT have also been verified. **Figure 10** shows an experimental load-management run for HTTP service, which we describe next.

HTTP Service

As shown in Figure 10, the clients generate HTTP requests to the advertised IP address/port-number combination (IP address `192.238.0.50` and Port `80`).

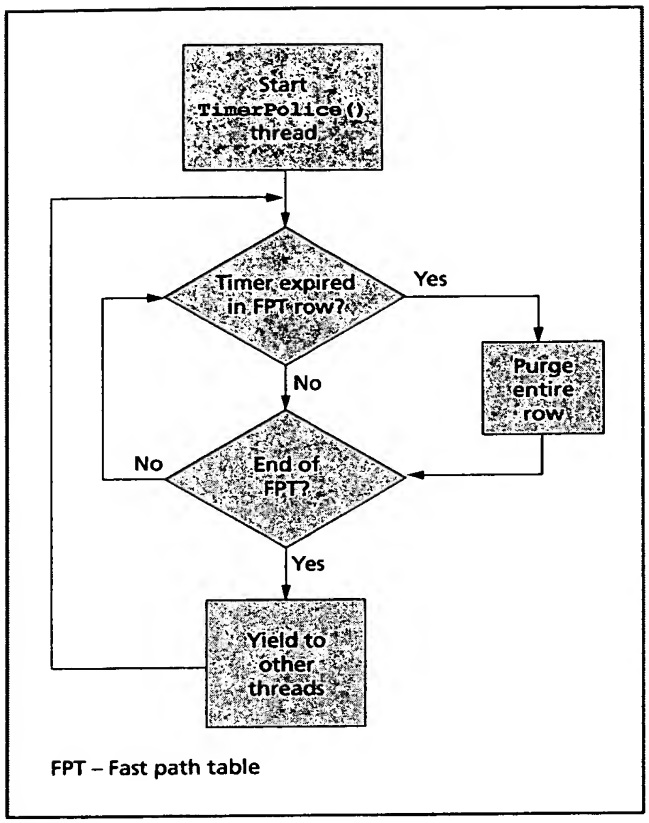

Figure 8.
`TimerPolice()` thread of the prototype.

The load-management prototype uses the FPT, the SPT, and the load-balancing algorithm to select a server from the farm. It is important to note that the SP identifies the type of connection as an HTTP request for a well-known port, and can, therefore, appropriately select a timer expiry value for the corresponding entry in the FPT. Irrespective of whether the connection request is an HTTP 1.1 (persistent) or HTTP 1.0 (non-persistent) type, the timer interval guarantees that any further requests generated from the same client will be directed to the same server. In the figure, the request from Client 1 happens to be sent to Server 2 by the load-management switch (after the application of the selection algorithm), as illustrated by the matching line styles of the *request* and *response* signals. The requests from Clients 2 and 3 are serviced by Servers 1 and 3, respectively.

It can be noted that all three clients sent their HTTP requests to the *same* IP address (`192.238.0.50`), the

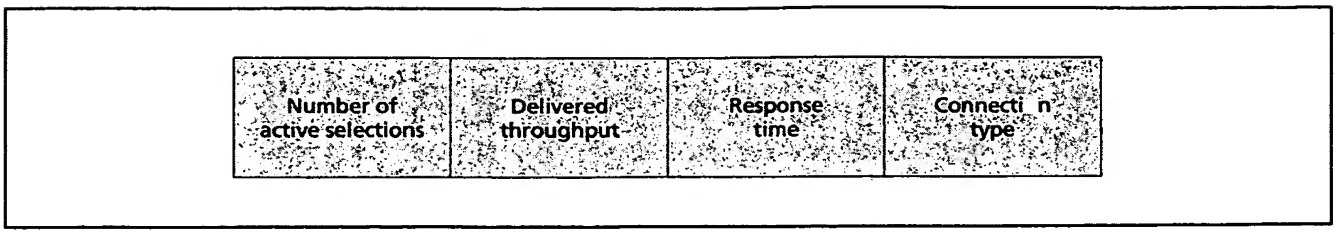

Figure 9.
Statistics collected by load-management switch for each farm server.

address advertised by the switch. This is also the aliased IP address all the servers are listening to, in addition to their own IP addresses. The switch successfully changed the source and destination MAC addresses of the Ethernet frames, which were then picked up by the intended servers in the farm.

FTP Service

Balancing FTP requests is an interesting case for our prototype. Not only the combination of the FTP control port (20) and the FTP data port (21) must be dealt with, but also the scenario of clients changing data ports for the same FTP connection needs to be addressed. First, the prototype ensures that it can recognize the control port and data port connections from the same client. This is achieved in the SP by looking at the client's IP address for a TCP packet with the FTP data port as the destination. The SPT is searched for an entry from the same client with the FTP control port as the destination. The same server selected earlier by the load-balancing algorithm for the control connection is then selected for the data connection as well. Second, if the client's data port number changes during an FTP connection, the keys used for an FPT or SPT lookup ensure that packets from the FTP client with the newer data port are sent to the previously assigned (correct) server. Therefore, it is not required to perform an SP update every time the client's data port changes for an ongoing FTP session. Multiple simultaneous FTP sessions from the client group have been successfully balanced by our prototype.

Multicast Video-on-Demand Service

Compressed video delivery through the prototype can be achieved smoothly with the load-management prototype. Two aspects of this type of traffic are important—the isochronous nature and the multicast aspect, where potentially many clients can send feedback/repair-request messages to the server. The isochronous nature (time sensitivity) can be dealt with using high-priority queueing, if necessary; otherwise, the bandwidth available on a high-end switch like the Cajun P550 switch may be sufficient for forwarding of the video packets with minimum delay.

At the IP level, the general structure of multicast packets remains unchanged. These packets are identified by a class D IP address[10] with the first nibble set to 1110. Thus, multicast IP packets have addresses ranging from 224.0.0.0 to 239.255.255.255. For an ongoing multicast video session from the server farm to a group of clients, it is possible that rate-control or repair-request feedback messages will be sent from some of the clients in the group to the server farm.[11,12] The IP packets of these feedback messages will contain the multicast IP address of the ongoing session. Using this field, the switch can identify that the packet is from an ongoing session and that it is not a new request to be balanced; the switch can, thereby, forward the packet to the appropriate server in the farm.

Conclusion

In this paper we have presented a fully functional load-management switch prototype that can be built into a commercially available Gigabit Ethernet IP switch offering from Lucent Technologies (such as the Cajun P550 switch). The prototype takes into consideration all the related key functionalities of the switch and demonstrates modules that will interwork with other existing hardware/software modules to route new incoming client requests to a group of servers connected to the switch. This paper also establishes a MAC-based address translation in order to operate correctly with the IP-aliasing feature available in modern

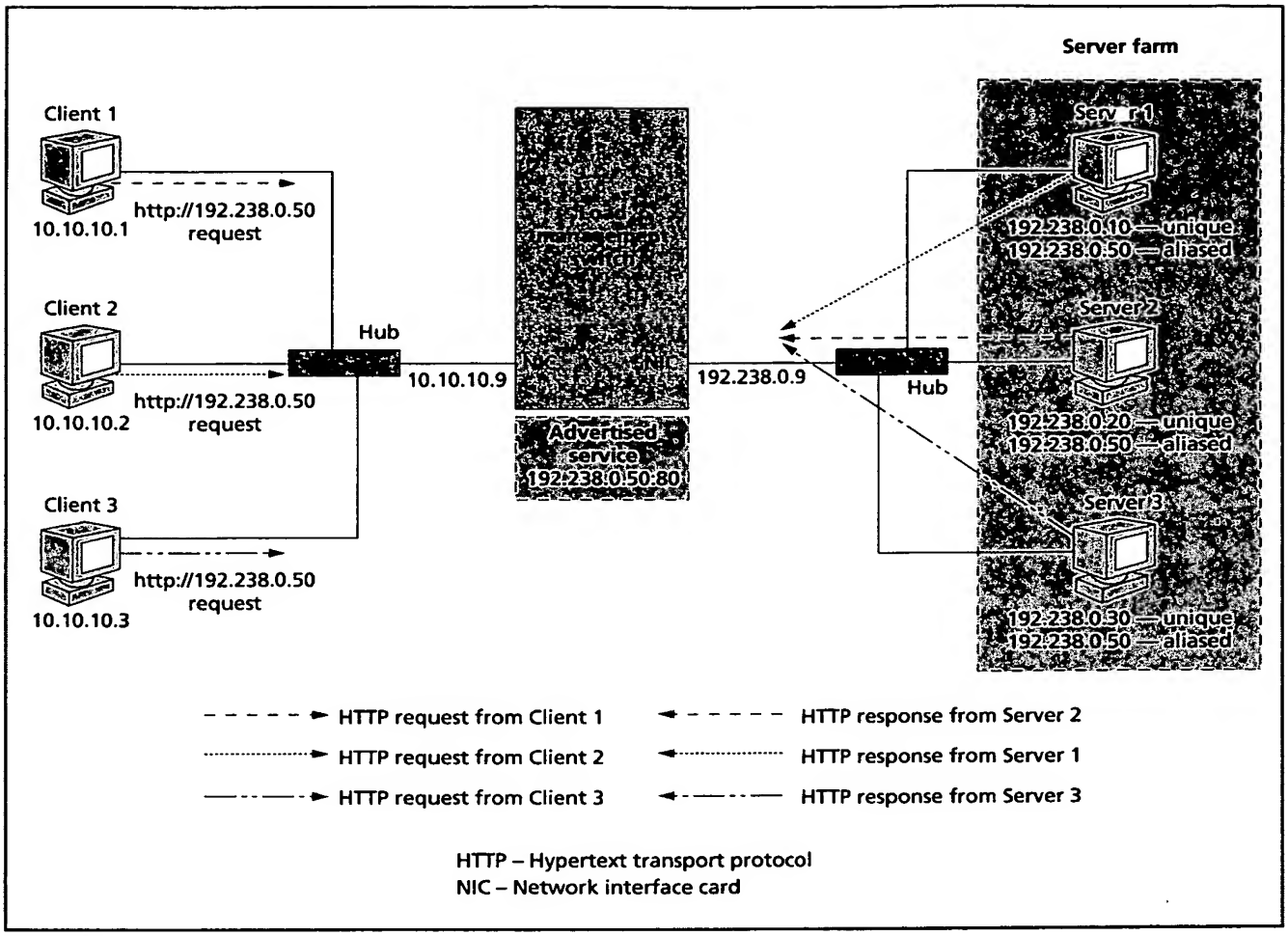

Figure 10.
Experimental run on load-management prototype.

OSs. A custom balancing algorithm that serves as the key to intelligent decision-making for Internet requests has been developed and described in this paper. This algorithm and the other features demonstrated in this prototype are flexible enough that they can be implemented in other network switches with reasonably similar architectures without major modifications.

***Trademarks**

Content Smart is a trademark of ArrowPoint Communications, Inc.

IRIX is a registered trademark of Silicon Graphics, Inc.

Linux is a registered trademark of Linus Torvalds.

Red Hat is a registered trademark of Red Hat, Inc.

Solaris is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark of The Open Group.

References

1. M. L. Stevens and W. J. Weiss, "Policy-Based Management for IP Networks," *Bell Labs Tech. J.*, Vol. 4, No. 4, Oct.–Dec. 1999, pp. 75–94.
2. Y. Bernet, J. Binder, S. Blake, M. Carlson, B. E. Carpenter, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, and W. Weiss, "A Framework For Differentiated Services," draft-ietf-diffserv-framework-02.txt, IETF, Feb. 1999, http://www.ietf.org
3. V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic Load Balancing On Web-Server Systems," *IEEE Internet Computing*, Vol. 3, No. 3, May–Jun. 1999, pp. 28–39.
4. D. Cartwright, "Bypassing the Traffic," *InformationWeek*, No. 53, Feb. 3, 1999, pp. 44–45.

5. B. Narendran, S. Rangarajan, and S. Yajnik, "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access," *Proc. of 16th Symp. on Reliable Distributed Systems, SRDS'97*, IEEE Comput. Soc., Oct. 1997, pp. 97–106.
6. O. P. Damani, P. E. Chung, Y. Huang, C. Kintala, and Y.-M. Wang, "ONE-IP: Techniques for Hosting a Service on a Cluster of Machines," *Computer Networks and ISDN Systems*, Vol. 29, No. 8-13, Sept. 1997, pp. 1019–27.
7. V. P. Kumar, T. V. Lakshman, and D. Stiliadis, "Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet," *IEEE Comm. Mag.*, Vol. 36, No. 5, May 1998, pp. 152–164.
8. Alteon WebSystems, Inc., "Next Steps in Server Load Balancing," White Paper, http://www.alteon.com
9. ArrowPoint Communications, http://www.arrowpoint.com
10. D. E. Comer, *Internetworking With TCP/IP, Vol. 1: Protocols and Architecture, 2nd Ed.*, Prentice-Hall, Englewood Cliffs, N.J., 1991.
11. S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing," *Computer Comm. Rev.*, Vol. 25, No. 4, Oct. 1995, pp. 342–356.
12. K. Nahrstedt, A. Hossain, and S.-M. Kang, "A Probe-based Algorithm for QoS Specification and Adaptation," *Proc. of 4th Intl. IFIP Workshop on Quality of Service (IWQoS'96)*, IFIP, Mar. 1996, pp. 89–100.

(Manuscript approved December 1999)

ASHFAQ HOSSAIN is a member of technical staff in the Data Communication Technology Department of Bell Labs in Murray Hill, New Jersey. He holds a Ph.D. in computer science from the University of Illinois at Urbana-Champaign. His interests include multicast-capable intelligent server/client development and protocol-independent switching over the Internet. Dr. Hossain is currently involved in designing a protocol-independent data-networking chipset to be incorporated into the most advanced switches from Lucent Technologies.

HOUSHING F. SHU is a member of technical staff in the Data Communication Technology Department at Bell Labs in Murray Hill, New Jersey. He has a B.S. degree in physics from National Taiwan University in Taipei and both M.S. and Ph.D. degrees in systems engineering from the University of Pennsylvania in Philadelphia. He was a recipient of the Internal Research and Development (IRAD) Program Manager and Author Award in 1989 and the Outstanding IRAD Performance Award in 1990. His current responsibilities include development of software for the Atlanta™ ATM Port Controller (APC) device driver, for which he received the 1999 Bell Labs President's Gold Award.

CHARLES R. GASMAN is a technical manager in the Data Communication Technology Department at Bell Labs in Murray Hill, New Jersey. He holds a B.S. degree in electrical engineering from Rutgers University in New Brunswick, New Jersey. He and his group specialize in the development of embedded software for high-performance IP- and ATM-based systems.

RANDOLPH A. ROYER is a member of technical staff in the Data Communication Technology Department at Bell Labs in Murray Hill, New Jersey. He has a B.S. degree in computer science from Rutgers University in New Brunswick, New Jersey. Currently he is developing software for advanced IP networking features in Lucent Technologies products.◆